3 ソフトクリーム (Softcream)

Author : 平田 誠治 (define)

小課題 1

この小課題では $X = Y = 1$ より選択の余地は Alice のトッピングのみである．したがって，すべてのトッピングを試して，最もスコアの高くなるものを選択すれば良い．
時間計算量は $O(Z)$ である．

```
int answer=0;
for(int c:C){
    int price=A[0]+B[0]+c;
    if(abs(price-P)>answer){
        answer=abs(price-P);
    }
}
```

小課題 2

この小課題ではトッピングに加えて，コーンにも選択の余地がある．両者が最善を尽くすという設定であるから，Bob はコーンの選択の時点で Alice がスコアを最大化するトッピングを選択する事を考慮しなければならない．
$g(b)$ を Bob が値段 b のコーンを選択した仮定のもと，Alice のトッピングの選択によって取りうるスコアの最大値とする．Bob は $g(b)$ が最小となるようなコーンを選択すれば良い．$g(b)$ の計算は，フレーバーとコーンを固定した場合を考えているので小課題 1 に帰着される．
j $(1 \leqq j \leqq Y)$ について $g(b)$ の計算に $O(Z)$ かかるから，全体の時間計算量は $O(YZ)$ である．

```
    int answer=(int)(3e8);
    for(int b:B){
        int g=0;
        for(int c:C){
            int price=A[0]+b+c;
            if(abs(price-P)>g){
                g=abs(price-P);
            }
        }
        if(answer>g){
            answer=g;
        }
    }
```

小課題 3

この小課題ではフレーバーにも選択の余地がある．小課題 2 と同様にして，Alice はフレーバーの選択の時点で Bob がスコアを最小化するコーンを選択する事を考慮しなければならない．

$f(a)$ を Alice が値段 a のフレーバーを選択した仮定のもと，Bob のコーンの選択によって取りうるスコアの最小値とする．Alice は $f(a)$ が最大となるようなフレーバーを選択すれば良い．$f(a)$ の計算は，フレーバーを固定した場合を考えているので小課題 2 に帰着される．

i ($1 \leqq i \leqq X$) について $f(a)$ の計算に $O(YZ)$ かかるから，全体の時間計算量は $O(XYZ)$ である．

このアルゴリズムは Mini-Max 法として知られるものである．

```
int answer=0;
for(int a:A){
    int f=(int)(3e8);
    for(int b:B){
        int g=0;
        for(int c:C){
            int price=a+b+c;
            if(abs(price-P)>g){
                g=abs(price-P);
            }
        }
        if(f>g){
            f=g;
        }
    }
    if(f>answer){
        answer=f;
    }
}
```

小課題 4

小課題 3 のアルゴリズムを高速化する事を考える.

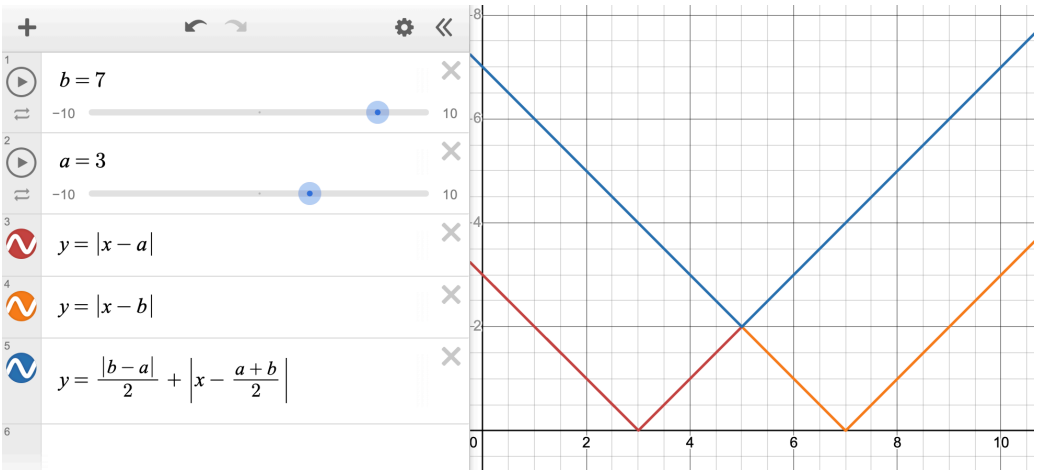
実は Alice のトッピングの選択は，値段が最小/最大のもののみを考慮すれば十分である．Alice の目標は値段の合計と P との絶対値であるスコアを最大化する事であるから，値段が最小のものと最大のもののうち，スコアがより高い方を選択すれば良い.

全体の時間計算量は $O(XY)$ である.

満点

小課題 4 ではトッピングの選択にかかる計算量を削減したが，ここではコーンの選択を高速化する．
フレーバーを値段 a に固定した際に，値段 b のコーンを選択した際の最終的なスコアは
$\max(|a+b+C_{min}-P|, |a+b+C_{max}-P|) = \max(|b-(P-a-C_{min})|, |b-(P-a-C_{max})|)$ である．

一般に $\max(|x-a|, |x-b|) = \frac{|b-a|}{2} + |x-\frac{a+b}{2}|$ であるから，
$\max(|b-(P-a-C_{min})|, |b-(P-a-C_{max})|) = \frac{C_{max}-C_{min}}{2} + |b-(P-a-\frac{C_{min}+C_{max}}{2})|$ となる．
イメージがつかない読者はグラフを参照されたい．



したがって，Bob は $P-a-\frac{C_{min}+C_{max}}{2}$ との差が最小となるようなコーンを選択すれば良い．これは事前に B をソートしておくことで二分探索で求められる．ソートにかかる計算量が $O(Y \log Y)$，各フレーバーについて二分探索にかかる計算量が $O(\log Y)$ であるから，全体の時間計算量は $O((X+Y)\log Y + Z)$ である．この解法を実装したのが C++ 実装例となる．

なお，$|b-(P-a-\frac{C_{min}+C_{max}}{2})|$ は b について単調減少関数と単調増加関数が連結した形であるから，最適な b の値の見当がつかない場合は三分探索を用いる事もできる．ソート後の B の index に対する三分探索を行う場合は，B の重複する要素を削除する必要がある事に注意されたい．
また，別解としてあるスコアが実現可能かどうかを二分探索する解法が挙げられる．フレーバーを固定した時に，あるスコアを達成するコーンが存在するかは二分探索で判定できるので，時間計算量は $O(X \log Y \log(A_{max}+B_{max}+C_{max}) + Y \log Y + Z)$ である．この解法を実装したのが Python 実装例となる．